How to release: Branching

Create a branch, if you are releasing a major version

This is done only once per major version (4.x, 5.x, 6.x, etc.) about 4-6 weeks before the planned official release of x.0. For minor versions, the work is done in current branch.

From a trunk checkout

```bash
php doc/devtools/svnbranch.php branches/20.x
```

After that, you need to change a few details in `lib/setup/twversion.class.php`, in both the New Branch and Trunk:

In the new branch, e.g. 20.x, change `$this-branch`. If already chosen, change `$this-star` from 'TBA' to the new star name

```php
// Set the development branch. Valid are:
//   stable   : Represents stable releases.
//   unstable : Represents candidate and test/development releases.
//   trunk    : Represents next generation development version.
$this->branch = 'unstable';
(...)
$this->star = 'Tarazed';  // 'TBA' if not chosen
```

Commit the above changes with a message such as "REL Update $this->branch and Star Name"

In Trunk, change `$this-version`, e.g. 21.0svn and, if available, add star name

```php
$this->version = '21.0svn';    // needs to have no spaces for releases
(...)
// if already chosen, add chosen star in function tikiStars()
31 => 'Tarazed',      // 20.x
```

Commit the above changes with a message such as "REL Change Trunk version to 21.0svn, added Star Name for 20.x"

More details on how to create a branch: [SVNTips#Handling_branches](#)

1.1.1. Post-branching operations

1.1.2. Make sure the star name picking process is almost over

This star name is needed for the alpha because it is in the path:
[https://sourceforge.net/projects/tikiwiki/files/](https://sourceforge.net/projects/tikiwiki/files/)

1.1.2.1. Update `$profilesLink`

- In `lib/setup/wiki.php`, update `$profilesLink` to the new branch
1.1.2.2. Update profiles.t.o site

- Create a new category for the new branch
- Add the category of the new branch to the wiki pages holding the profiles listed in the Profiles Wizard, unless you know that they shouldn't work with the new tiki branch for some reason (deprecated features involved, etc)
- Contact the Profiles Team to report if any profile needs an update for some reason, or if you want to contribute a new profile for the new branch.

1.1.2.3. dev.tiki.org

- Create the new branch page (if not there yet), i.e.: Tiki18
  - Move the current alias from the old branch to the new created one (I'm not sure what that means?)
- Add the branch as an option for the Tiki Version field in the Bugs & Wish list tracker (category)
- Update Daily Build (You need to ask Oliver Hertel to add new versions)
- Update Get code
- Update Where to Commit
- Update Semi-automatic-merging-period (Not clear what's there to update)

1.1.3. packages.tiki.org

**Note:** The assets (img and css) supports up to 3 versions (trunk, 19.x and 18.x).

Update versions list (not appending more versions), example change to "trunk, 20.x and 18.x":

- Checkout the repository git@gitlab.com:tikiwiki/tiki-packages-build.git
- In .gitlab-ci.yml update the VERSIONS variable to `VERSIONS: "trunk 20.x 18.x"
- Commit to git master branch and wait for the build to complete.

Update to add a new version, example change to include 20.x version, "trunk 20.x 19.x 18.x"

- Checkout the repository git@gitlab.com:tikiwiki/tiki-packages-build.git
- In .gitlab-ci.yml update the VERSIONS variable to "VERSIONS: "trunk 20.x 19.x 18.x"
- In resources/assets/css/styles.css duplicate "VERSION 03" section to "VERSION 04" and change the colors.
- In resources/assets/img duplicate down-03.svg to down-04.svg
  - Open the svg file, change the title to down-04 and the stoke color to use the same color set in styles.css file.
- Commit to git master branch and wait for the build to complete.

Tiki Packages in GitLab: https://gitlab.com/tikiwiki/tiki-packages-build
Tiki Packages website: https://packages.tiki.org


1.1.4. show.tiki.org

First on the server:

- On the show server checkout the new branch into `usr/local/src/tiki`
- On the show server add the branch to `/usr/local/sbin/tim-common BRANCHES="trunk 12.x 13.x"`
- Refresh the instances by calling right away the cron: `/usr/local/sbin/tim-cron`
- Once dev.tiki.org updates, it should then work.

Then on the ShowTikiOrg tracker field options:

- Go to the tracker field admin (as a tracker administrator)
- Add the new branch in the "Supported Versions" option
- Clear the caches

1.1.4.1. demo.tiki.org

- Create a site for new version

1.1.4.2. Pre-dogfood servers

- Each Pre-Dogfood Server should be moved from trunk to the next branch and each site should go through at least 30 minutes of manual testing of the most common operations.
  Ex.: on `http://nextdev.tiki.org`, someone should try to report a bug, or on `http://nextdoc.tiki.org` try and add a new page to a structure (even though it all will be overwritten the next day).
  - the instances need to `svn switch` and refreshed immediately, e.g.
    + `svn switch https://svn.code.sf.net/p/tikiwiki/code/branches/13.x`
    + `bash setup.sh -n fix`
    + `php console.php database:update`
  - `/usr/local/bin/refresh-nxt.sh` needs to be updated so the next cron switches to the right branch
  - A couple of days before the release the cronjob to do a full upgrade including DB-sync from live sites must be disabled so that designers and gardeners can test/learn/prepare for the upgrade of the live community sites.
  - After release the `/usr/local/bin/refresh-nxt.sh` needs to be changed to trunk again and the full upgrade cron job reenabled.

1.1.4.3. Dogfood release policy

See dogfood for general background info on this policy. The general principle is that most of *.tiki.org sites should be running supported versions **before** they are released while others will keep running with previous version (LTS).

Goals:

- Reduce the number of issues and collective time spent on these. Issues can be bugs, data corruption, upgrade bugs (that you don't see in a fresh install), etc. in released versions.
- There is also a promotional goal for this policy as it reassures users that releases have sufficient testing, and it differentiates us from extension-based systems, where the official sites are upgraded to the latest versions a long time after dot zero.
• Improve the quality / reliability of .0 releases so we can eventually be in a position to shorten the release cycle. Historically, .0 releases have been shaky and too many people where waiting for .1 and that delays everything.
• Keeping some website running old LTS version is useful in case we need to produce and test security patch or any other correction necessary.

The various *.tiki.org sites collectively have a lot of users & data and cover quite a few features. Power users of these sites are usually quite familiar and can spot a bug or regression (something that stopped working) and report it efficiently.

Some sites are kept at the latest stable version (ex.: doc.tiki.org), while others are kept on the Long Term Support (LTS) versions (ex.: profiles.tiki.org). The list is maintained at Domains.

All sites are generally updated daily with minor revisions in each branch. So if a site is running 9.x LTS, it has the latest pre-released 9.x code. Exception: some legacy sites are kept for historical purposes in old unsupported versions, normally in read-only mode.

When a new major version is coming out, each of the sites identified as latest stable are progressively updated to the new version.

Start with the less critical one (ex.: themes.tiki.org)

1. Do a thorough check on the Pre-Dogfood Server version (each site should go through at least 30 minutes of manual testing of the most common operations. Ex.: on dev.tiki.org, someone should try to report a bug.)
   ○ The pre-dogfood servers generally run trunk, but during the pre-upgrade period, they should be switched to the branch
2. Fix all the bugs and check they are indeed fixed there
3. Proceed to upgrade (svn switch)
   ○ Keep previous version available as a snapshot so users can compare and efficiently report any upgrade issues (ex.: legacydev11.tiki.org)
4. Give a few days to get feedback for newly uncovered issues
5. And start over the cycle for each sites.

This whole process typically takes 3-5 weeks because of discovery of new issues and the understandable delay to resolve them. Because of this incompressible time factor, it is important to start the dogfood process as soon as possible in the release process (as soon as all bugs are resolved on the pre-dogfood server)

Finding & fixing an issue while in pre-dogfood avoids wasted time and corrupted data on the real sites. Trying to save time by rushing a release just creates more work down the line.

Related
• How to release